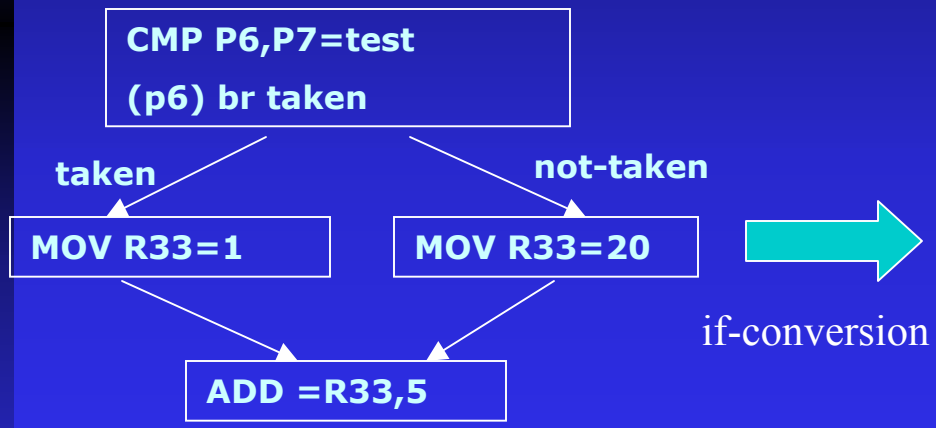
Phi-Predication for Light-Weight If-Conversion

Weihaw Chuang
Brad Calder
Jeanne Ferrante

University of California, San Diego

Benefits of If-Conversion

- Eliminates hard to predict branches
  - Important for deep pipelines
- How? Executes all paths with qualified update
  - False path- no updates nor sides-effects




```
CMP P6,P7=test ;;
(P6) MOV R33=1
(P7) MOV R33=20 ;;
ADD =R33,5
```

e.g. if P6=1, P7=0
R33 at ADD is 1

Predication Challenge

- Objective: Apply predication benefits to aggressive Out-Of-Order Execution
  - Why? Deep pipelines ⇒ Long branch misprediction penalty
  - If-Conversion
- Challenge: Multiple Definition Problem
  - Predication causes unnecessary stalls during Out-Of-Order register renaming
- Solution: ALWAYS-WRITE Phi-Predication.

Conditional-Writer Predication

- Today's predication is Conditional-Writer
  - Qualify state update and side-effects via predicate
    (P6) MOV R33=1 ⇒ **if(P6=T) then R33=1**
  - e.g. IA64, TriMedia, ARM

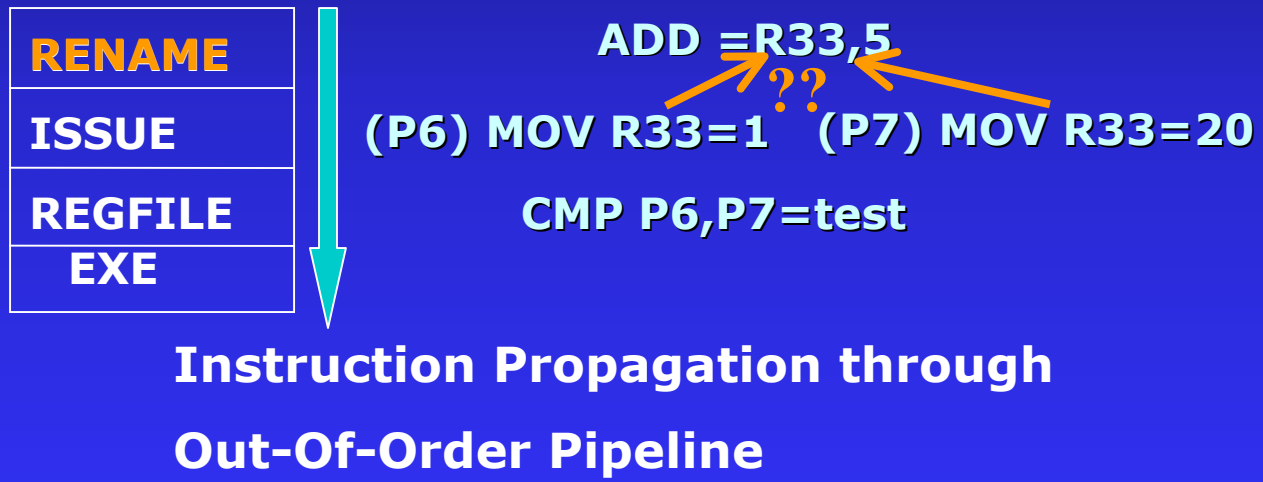
- Typically heavy-weight ⇒ Implementation cost is high
  - Most instructions qualified
  - Opcode space cost
  - Needs predicate-aware compiler analysis

Outline

- **Multiple-Definition Problem**
- Light-Weight Phi-Predication
- Code Generation
- Performance Result

Multiple Definition Problem

- Conditional-writer on Out-of-Order execution
- In OOO Register renaming
  - If predicate not ready ⇒ ambiguity choosing last def
- One Solution: Stalling till predicate ready ⇒ very costly



Outline

- Multiple-Definition Problem
- **Light-Weight Phi-Predication**
- Code Generation
- Performance Result

Phi-Predication ISA

- PHI

  phi r32=(p1),r33,r34

  phi r32=(p1),100,r34

  fphi f32=(p1),f33,f34

- MEMORY

  ld r32=[r33],(p1)

  st [r32]=r33,(p1)

- ORP

  orp p1=p2,p3,p4

- UNCONDITIONAL COMPARE

  cmp.eq.unc p1,p2=r3,r4,(p3)

  cmp.gt.unc p1,p2=r3,r4,(p3)

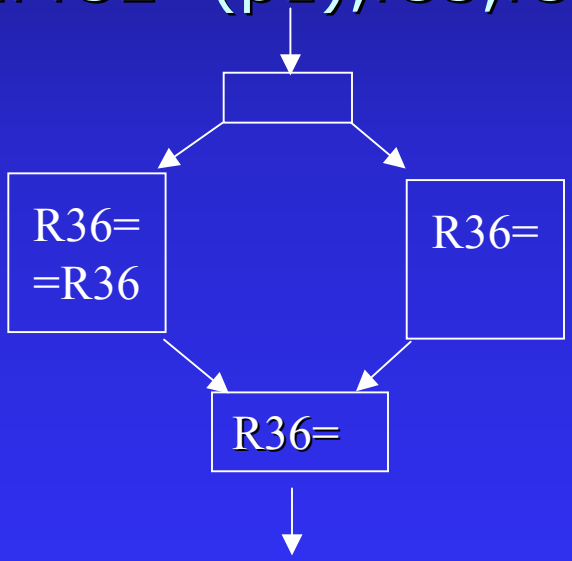
  cmp.ge.unc p1,p2=r3,r4,(p3)

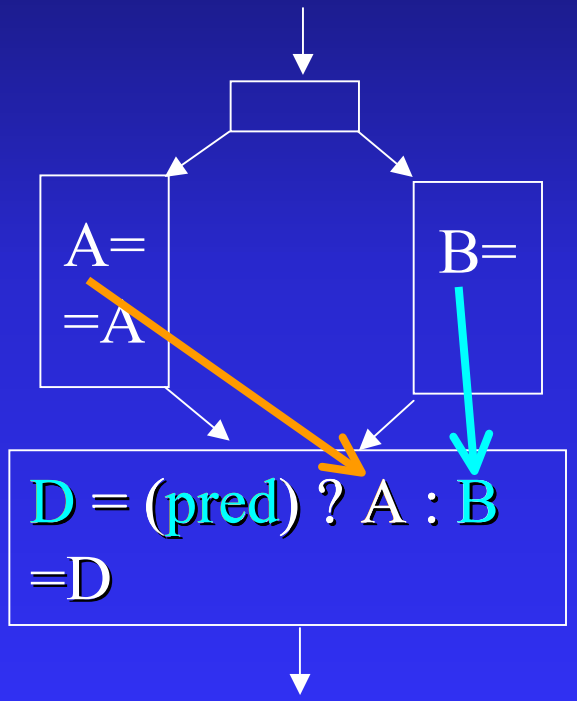
Phi-Instruction

- Key idea- Always-assign register destination
- Qualify register dataflow via selection
  - From Select operation (Multiflow ISA)
  - Regular compiler analysis

e.g.

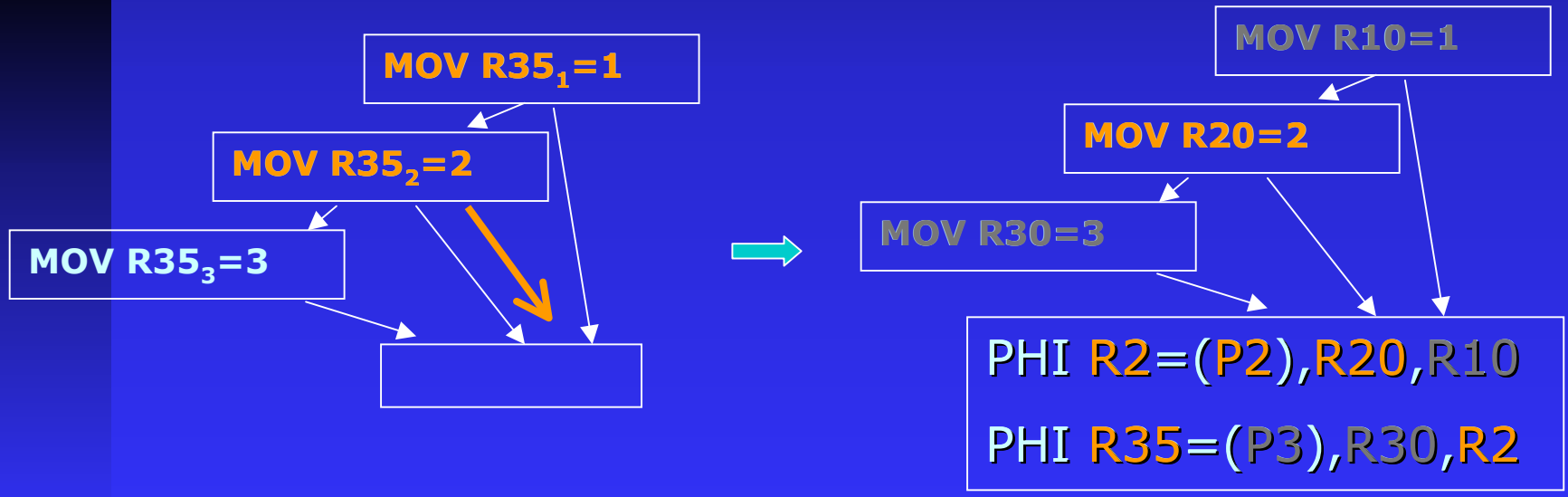
```
phi r32=(p1),r33,r34
```





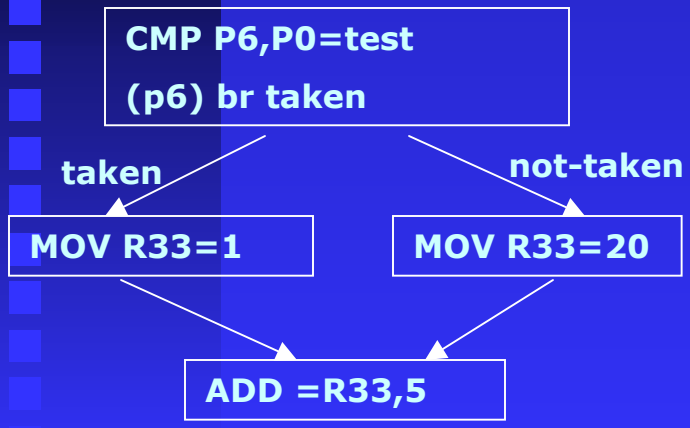
Phi Evaluation Chain

- Multiple join edges ⇒ maybe multiple PHI's
  - Rename definitions
  - For N definitions at join blk need N-1 PHI's to select correct definition
  - Evaluate phi's OPERANDS in topological order



Small Region If-Conversion

- Emphasize benefit of mispredicting branch removal
  - against penalty of executing all paths
- Small Regions
  - Best captures benefit
  - Similar schedule for branch, conditional-writer, Phi-Predication



Cond-Writer Predication	Phi Predication

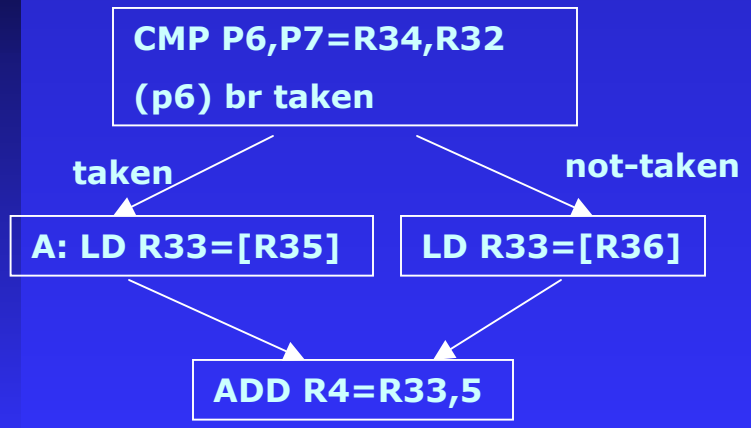
CMP P6,P7=R34,R32
;;
(P6) MOV R33=1
(P7) MOV R33=20
;;
ADD =R33,5
	MOV R35=1
	CMP P0,P7=R34,R32
	;;
	PHI R33=(P7),20,R35
	;;
	ADD =R33,5

If-Conversion Schedule Issues

- Avoid execution resource constraints- execute all paths lengthens schedule
- Avoid imbalanced paths- might penalize short path
- Avoid long Phi-Chains- long evaluation sequences caused by multiple join edges

Qualifying Memory Operations

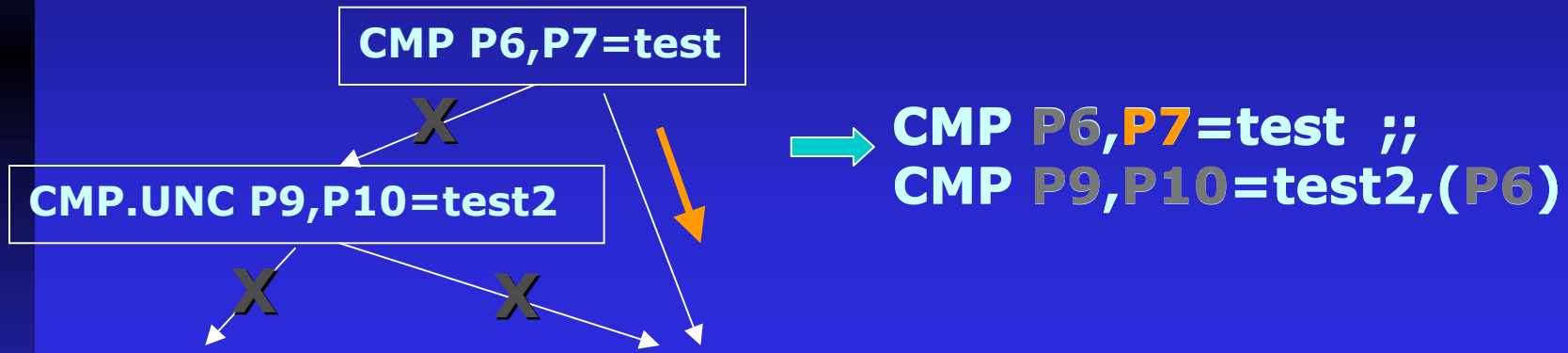
- Loads and Stores
  - Qualified exception and memory update
  - False loads assign destination value 0- obey always-write




```
CMP P6,P7=R34,R32
;;
LD R37=[R35],(P6)
LD R38=[R36],(P7)
;;
PHI R33=(P7),R37,R38
;;
ADD R4=R33,5
```

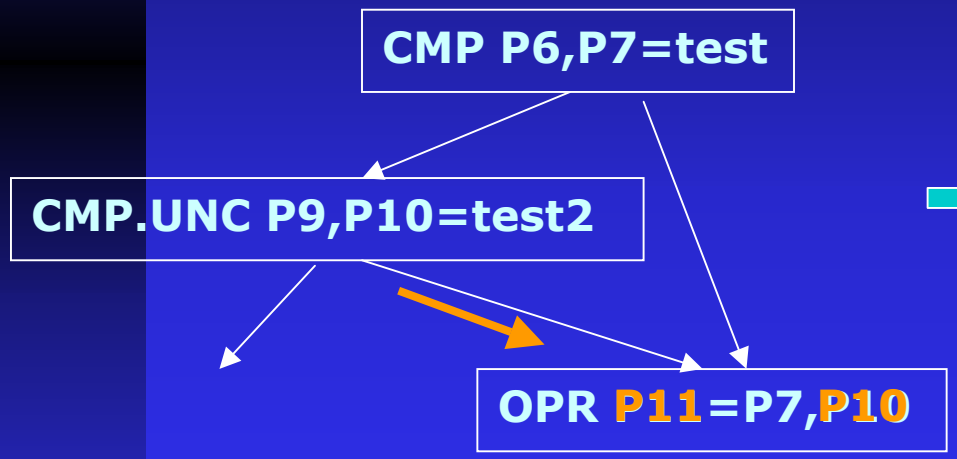
Qualifying Predicate Generators

- Unconditional Compares- (from IA64) for nested control flow
  - If qualified false ⇒ must not set predicates true



Generating Join Block Predicates

- OR'ing of predicates (ORP)
  - Predicates must obey always-write principle

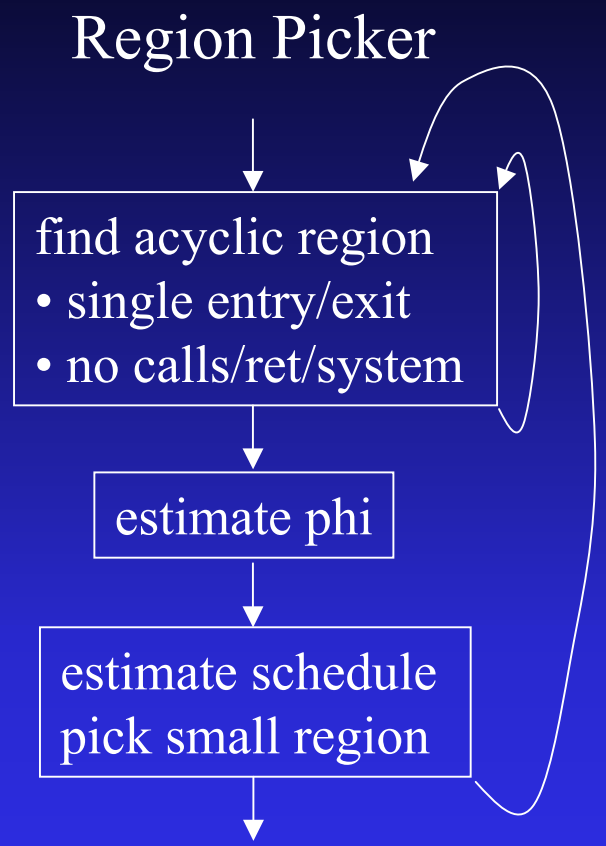



```
CMP P6,P7=test  ;;
CMP P9,P10=test2,(P6) ;;
ORP P11=P7,P10
```

Outline

- Multiple-Definition Problem
- Light-Weight Phi-Predication
- **Code Generation**
- Performance Result

Compiler Steps

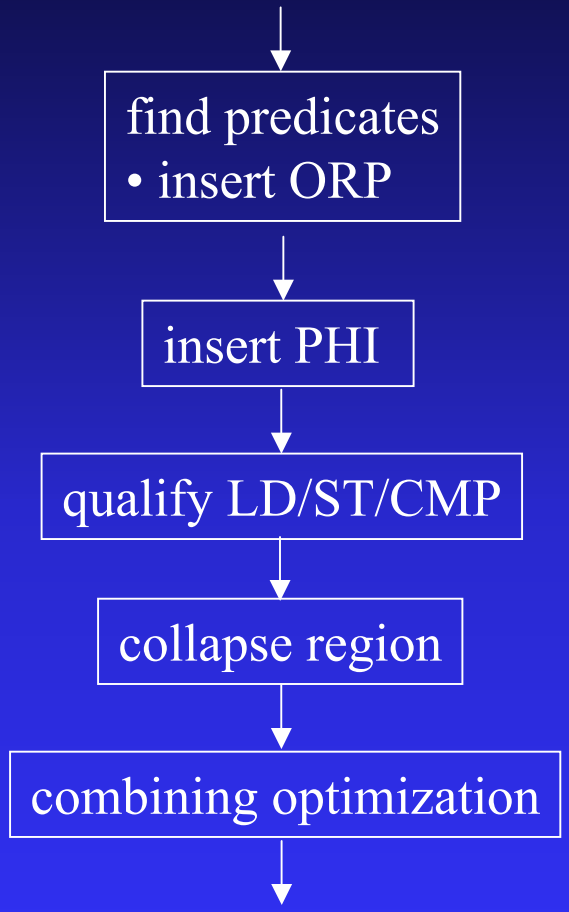


- Don't qualify expensive instructions or side-exit branches
- Determine schedule
  - estimate Phi insertion points

Modified Intel's IA64 compiler

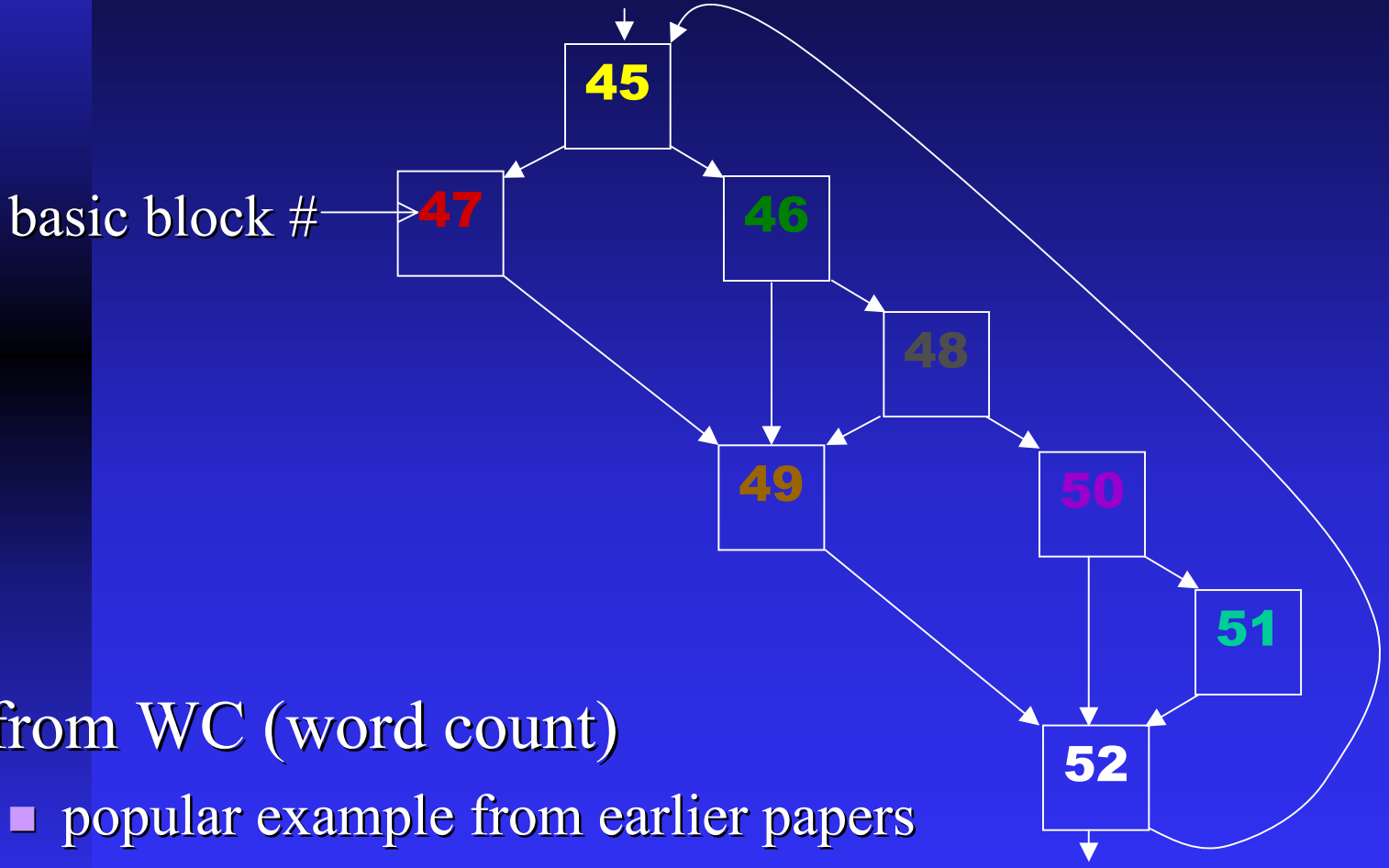
Compiler Steps

Code-Generator



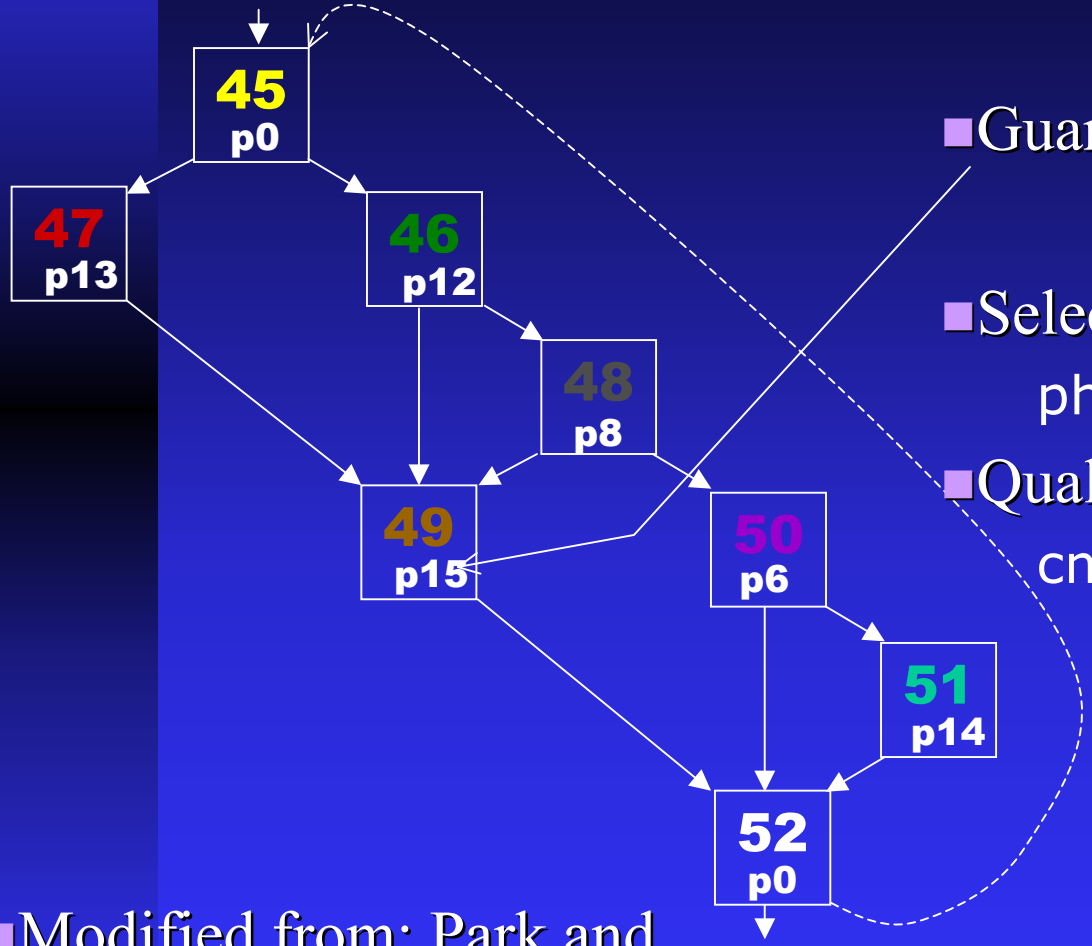
- Insert predicate generators
- Qualifies register data-flow
- Qualifies memory side-effects and predicate data-flow
- Delete branches, fold all path
- Delete unnecessary moves

Code Gen Example



- from WC (word count)
  - popular example from earlier papers

Guard Predicate



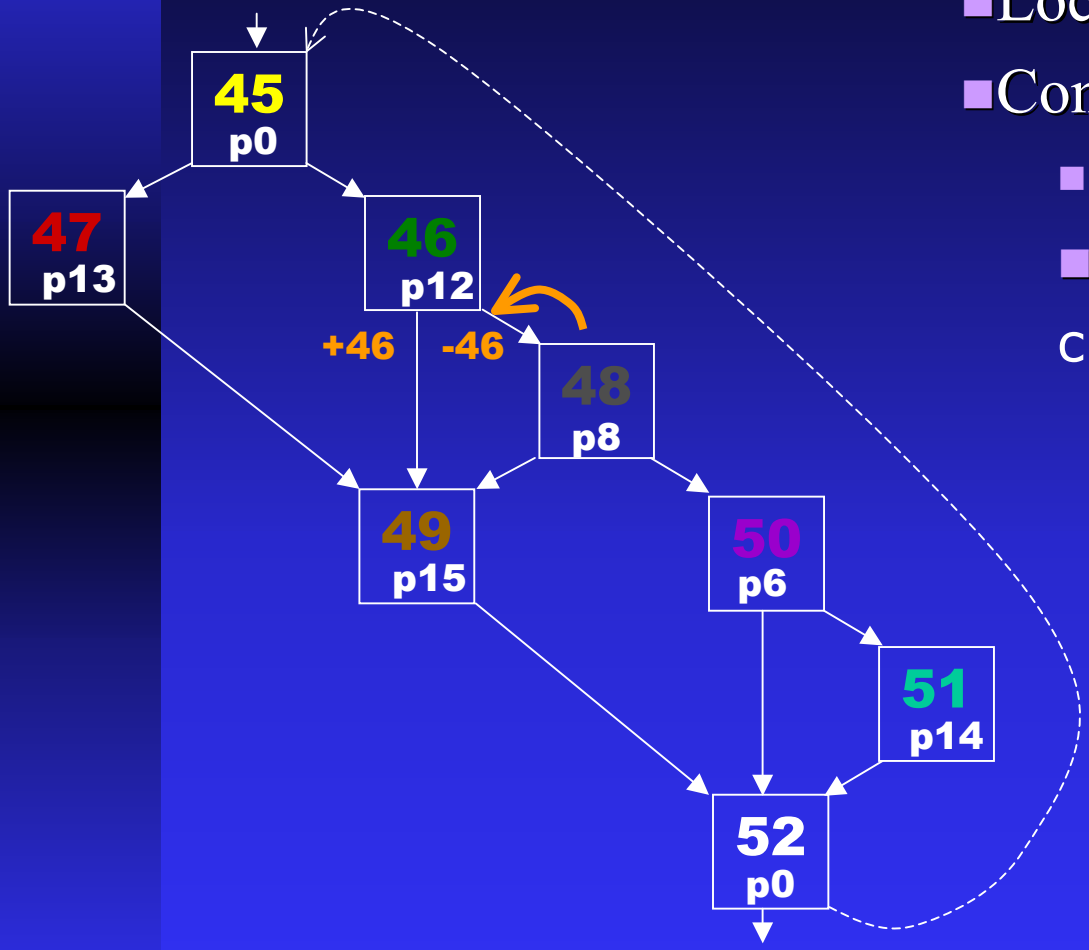
- Guard predicate
- Selection:

  phi r22=(p15),1,r38
- Qualification:

  cmp.unc p7,p6=32,r28,(p8)

- Modified from: Park and Schlanksker (RK Algorithm)

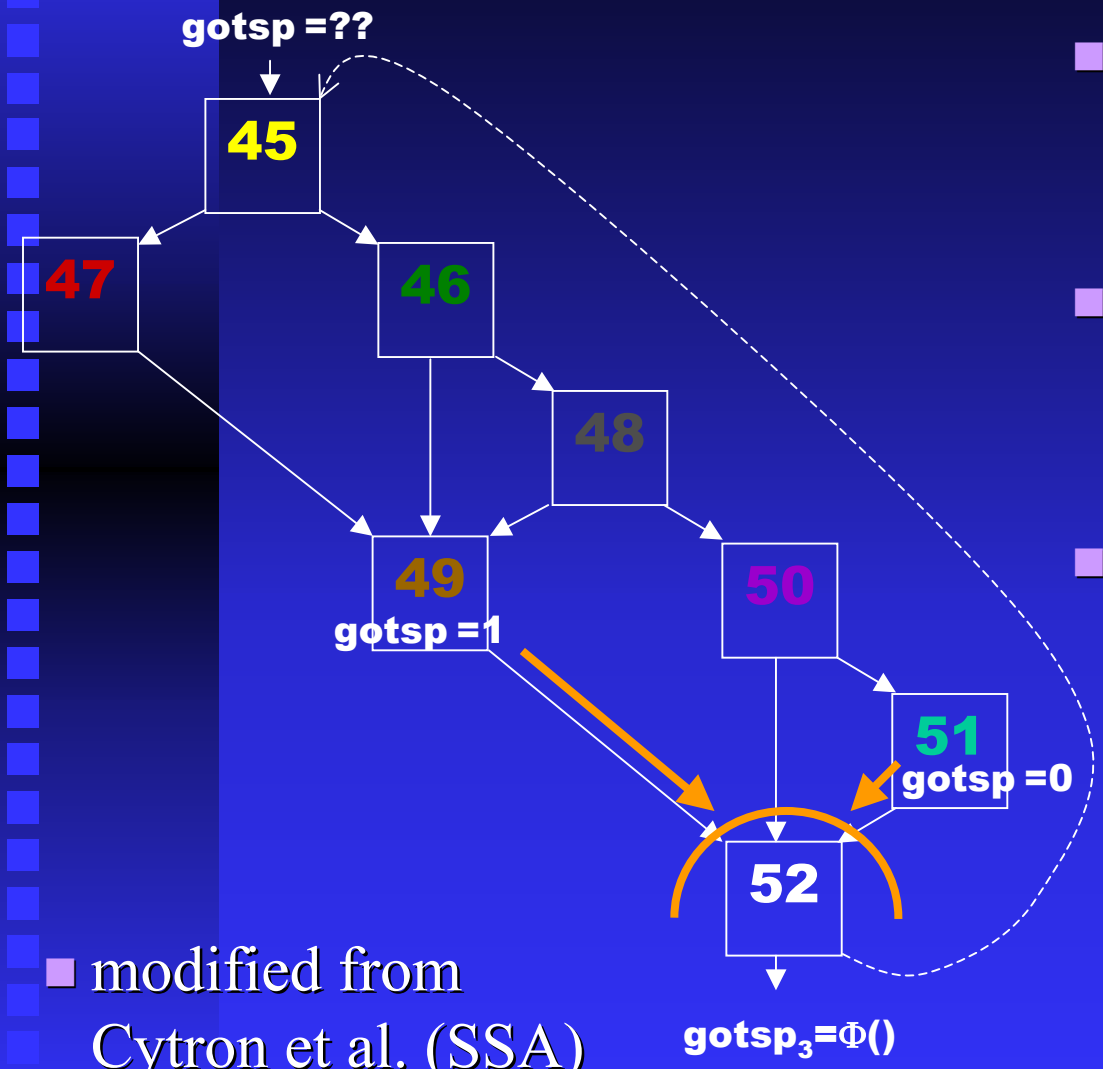
Predicate Generation



- Locate predicate generators
- Control Dependence
  - B46: CD(B48)={-B46}
  - Gen P8 at B46, neg edge:

cmp.unc -,p8=test,(p12)

Phi Insertion Point



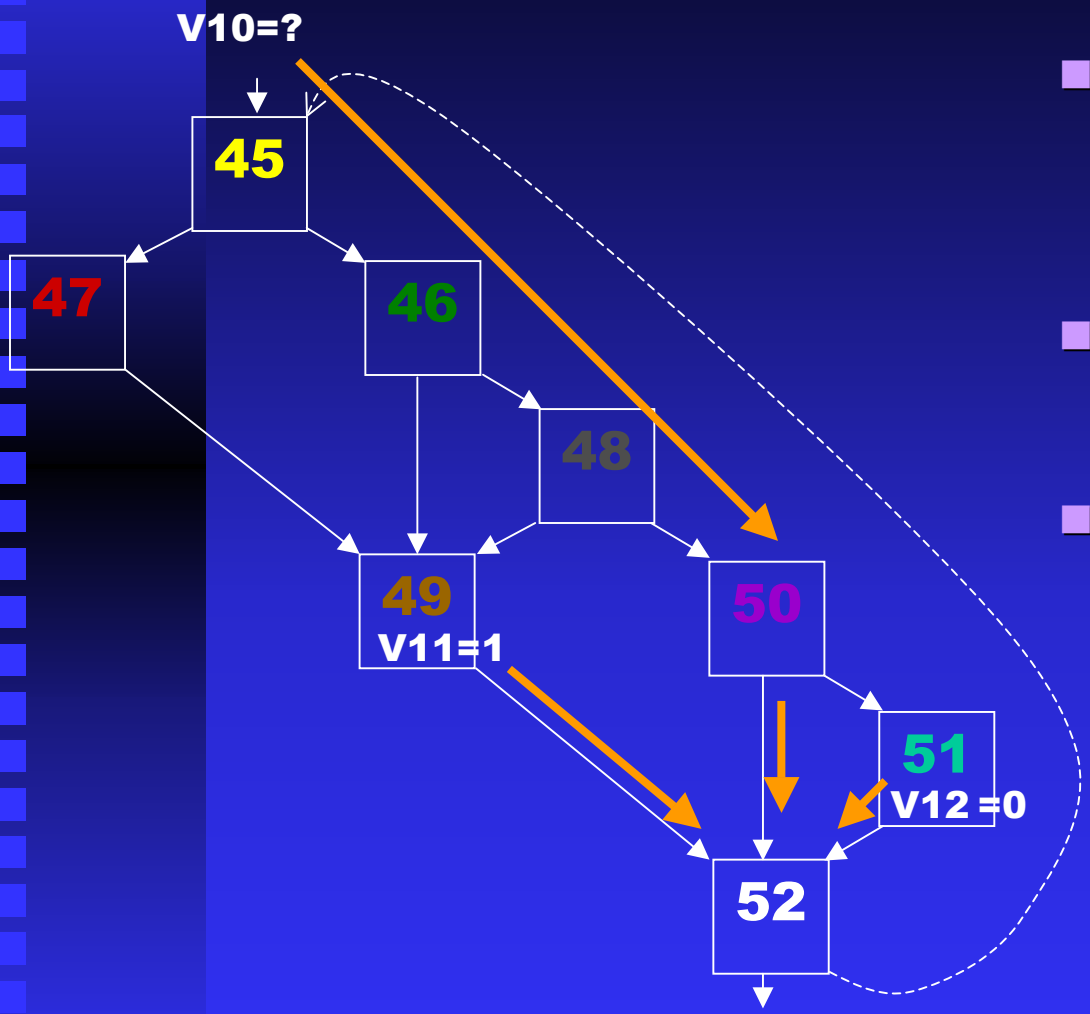
- Find insertion point Modified Static-Single Assignment (SSA)
- Dominance Frontier (DF)
  - Register dataflow decision point(s) at join
- Ex.

  DF(B49)={B52}

  DF(B51)={B52}

- modified from Cytron et al. (SSA)

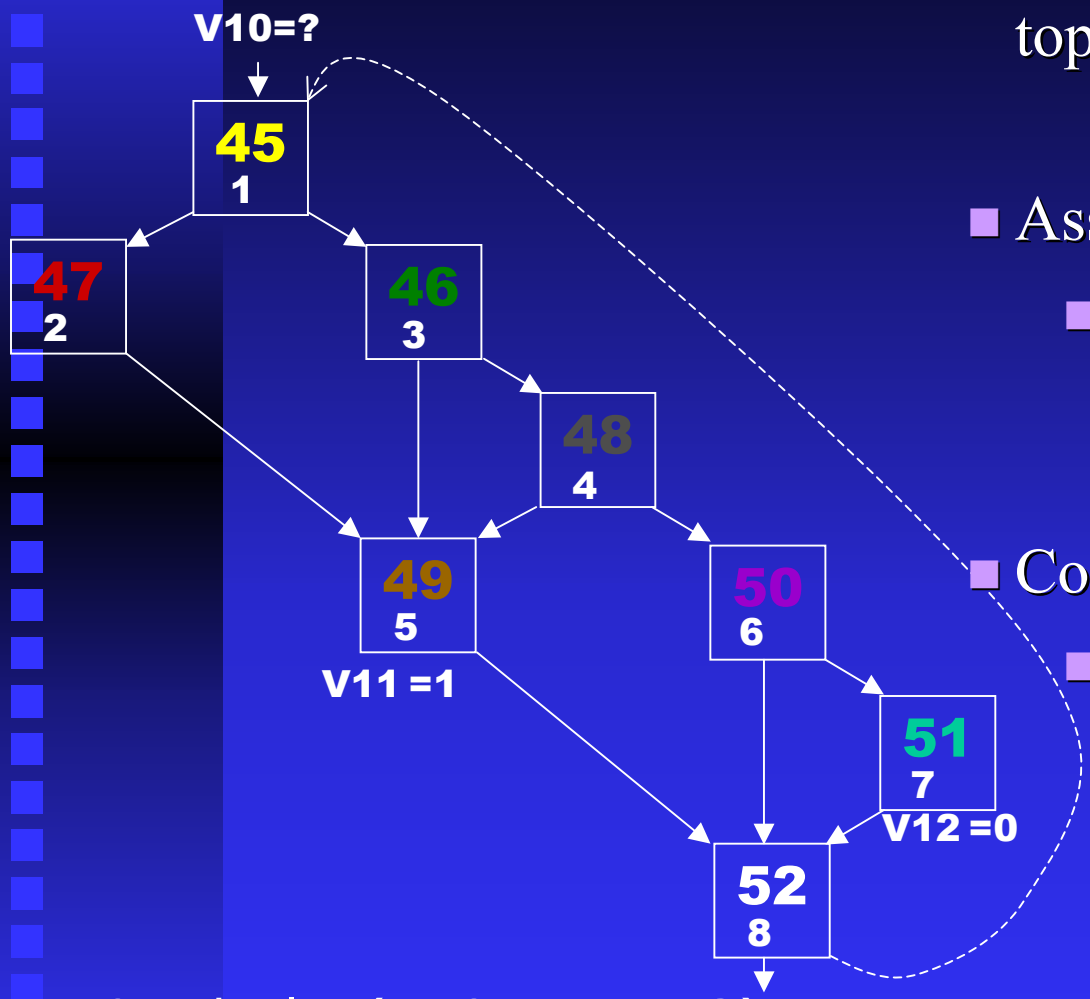
Unique Register Definitions



- Rename all assignments to unique instances in region
- Forward propagate to Uses
- Collect in phi-lists (at DF pt)

V10 = phi-list(V10, V11, V12)

Phi Code Generation

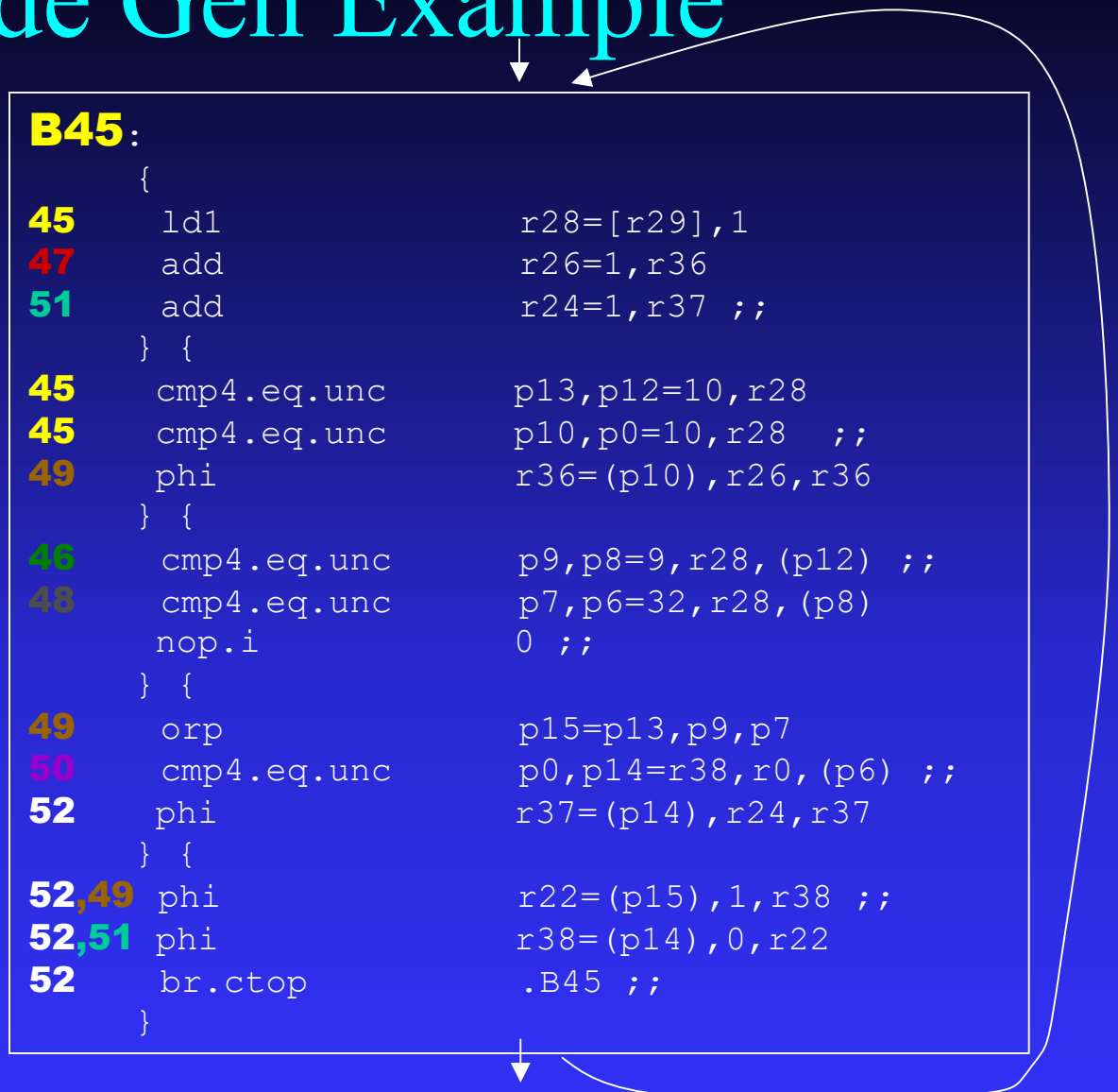


V10=phi-list(V10,V11,V12)

- Phi-List operands sorted topologically
- Associated w/operand
  - Def-block guard predicate ⇒ selector predicate
- Code gen: picks operand pairs
  - Use more specific selector predicate

V13= phi (p15) V11, V10
V10= phi (p14) V12, V13

Code Gen Example



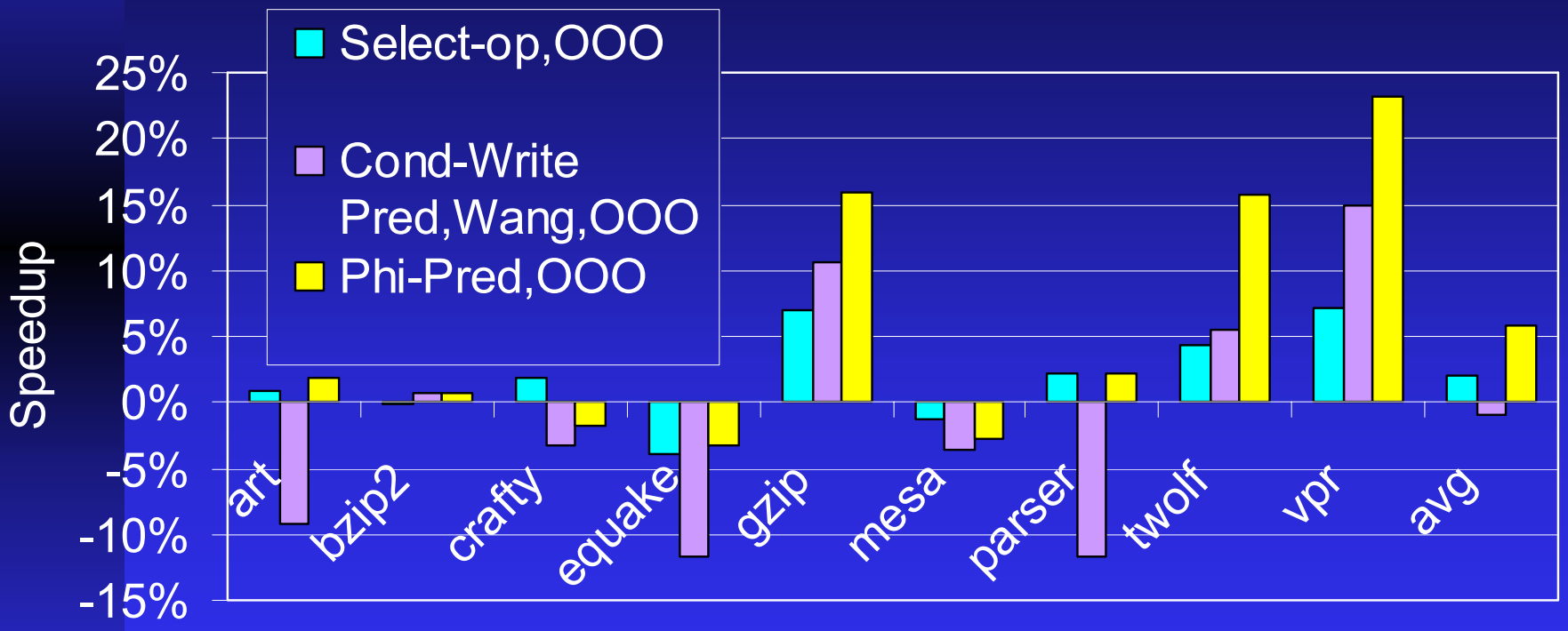
Outline

- Multiple-Definition Problem
- Light-Weight Phi-Predication
- Code Generation
- **Performance Result**

Simulation Models

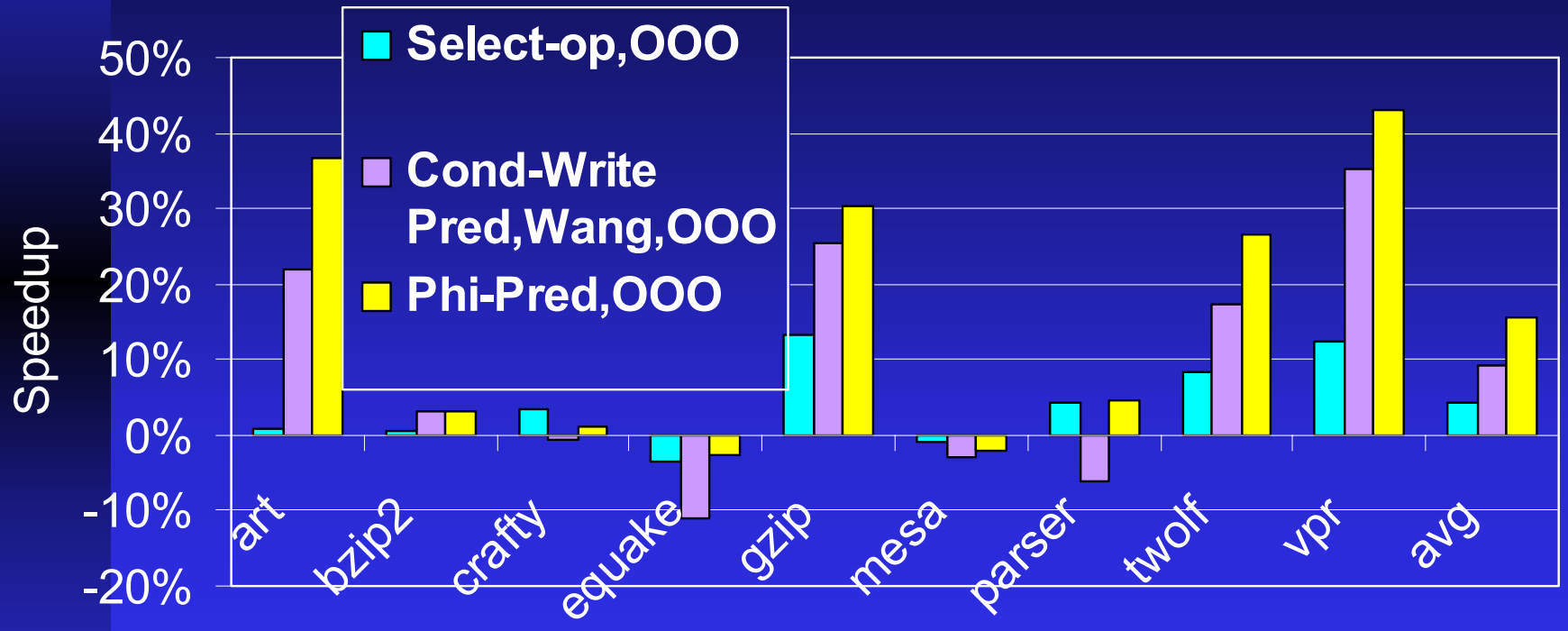
- "Select-op"
  - CMOV-like
  - allow only PHI, CMP.UNC
- "Wang et. al."
  - IA64: conditional-writer, heavy-weight ISA
  - RAT generated select-ops to avoid multiple definition problem
- "Phi-Op"
  - This paper
  - PHI, CMP.UNC, ORP, LD/ST

OOO Speedup (12 cycle penalty)



Normalized against no predication

OOO Speedup (30 cycle penalty)



- Sprangle and Carmean (deep pipelines)

Conclusion

- Future aggressive OOO processors will have very deep pipelines
- If-conversion helps reduce deep pipeline branch misprediction penalties

- **Phi-Predication** provides a light-weight implementation that solves the multiple-definition problem
  - described compiler Phi-Predication if-conversion

Backup Slides

- ISA Comparison
- More Code-Generation Tutorial
- Multiple-definition problem does not affect memory qualification

Comparison

Conventional:

- Heavy-Weight: Most instruction qualified
- OOO implementation has multiple-definition problem
- Predicate def-use dependency
- Requires predicate aware compiler analysis

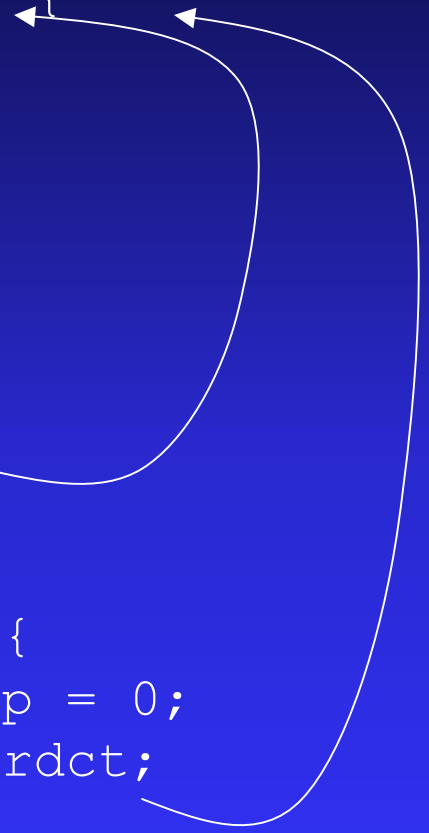
Phi-Predication:

- Light-weight:
  - four instruction types
  - phi, ld/st, cmp, ORP
- Always write definition
- Predicate promotion
- Regular compiler analysis

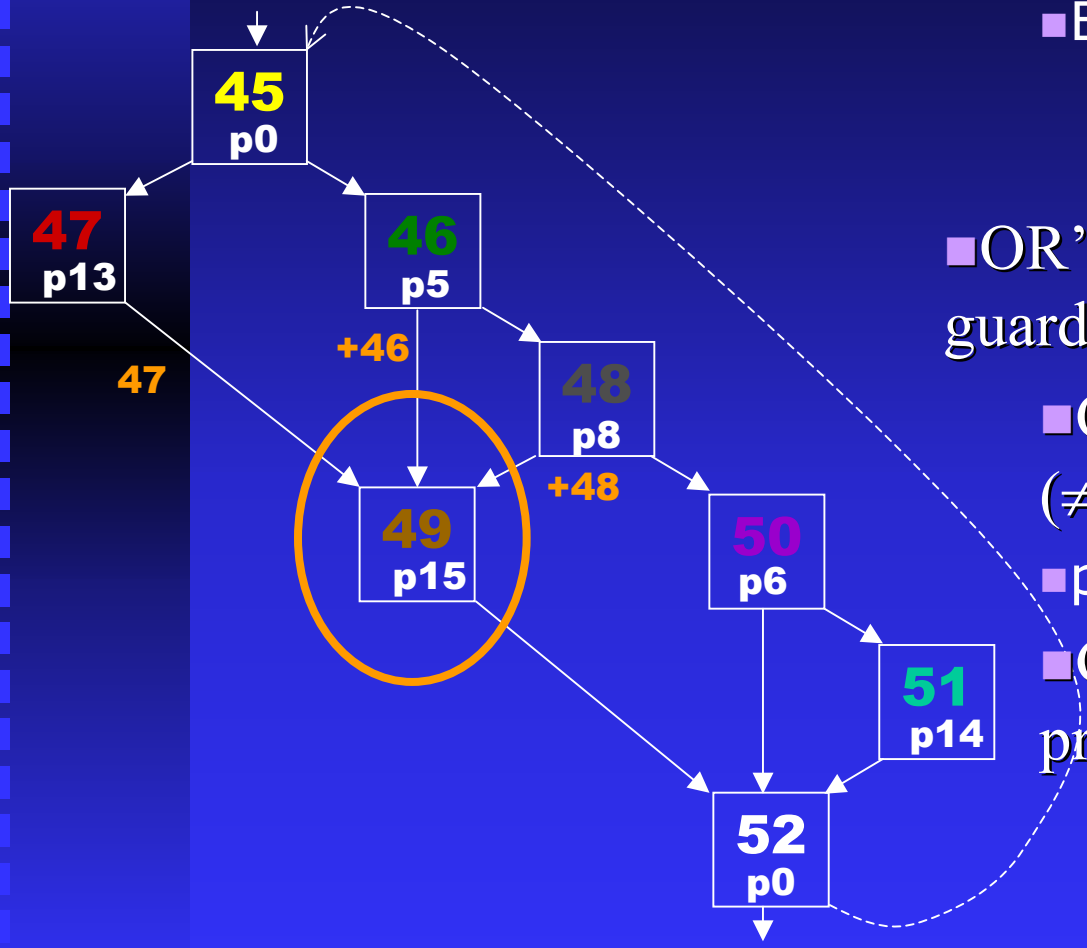
Code Gen Example: Inner Loop of cnt/wc.c

```
45,52 for(c=buf; len--; ++C) {
            switch(*C) {
45          case NL:
47                ++linecnt;
46          case TAB:
48          case SPACE:
49                gotsp = 1;
                  continue;

            default:
50                if(gotsp) {
51                      gotsp = 0;
                        ++wordct;
                  }
            }
}
```

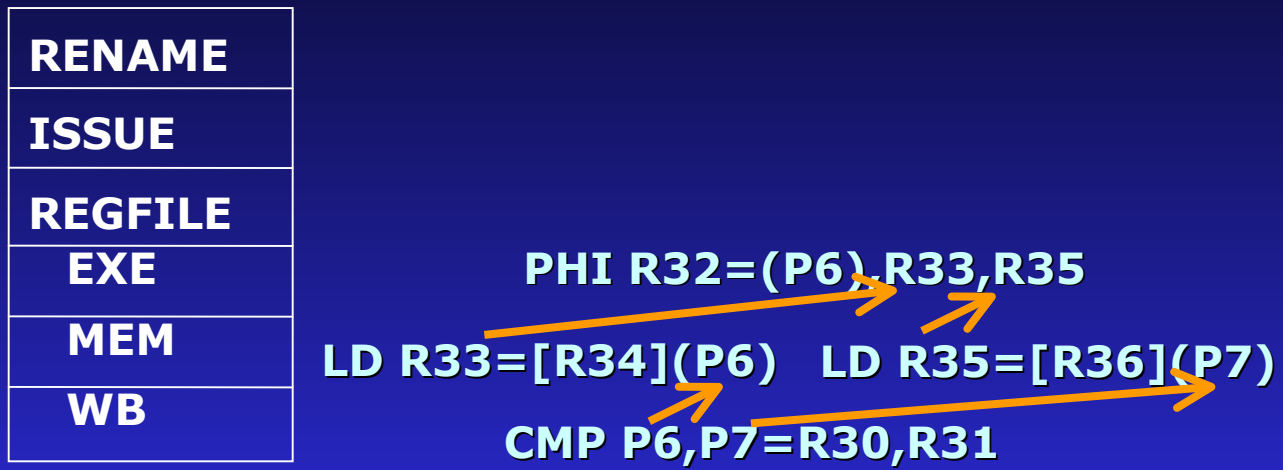


Predicate Generation: Multiple Incoming Edge



- Control Dependence (CD)
  - B46: CD(B49)= {B47,+B46,+B48}
- OR'ing control predicates to get guard predicate
  - Control predicate guards edge (≠ guard predicates!)
  - p15=ORP(p13,p9,p7)
  - Gen Uncond Comp for control predicate

Qualified Memory Update



- Ld/St memory multiple-definition problem not problematic
  - Typically predicates are ready